Apostila de MAC2166 - Aula 5

Autores:Beatriz Passanezi
Douglas Nicihoka

Oi gente! Na aula de hoje nós vamos falar sobre:

- Como trabalhar com números reais em C
- Leitura e escrita de arquivos de texto

Números reais em C

Antes de qualquer coisa, vamos explicar um pouquinho o modo que o computador armazena e interpreta esses números. Um número real é representado na maioria das vezes com uma notação parecida com a notação científica que vocês devem ter estudado no colegial. Por exemplo, o número 123.45 pode ser representado como 1.2345×10^2. O computador lê algo bem parecido, mas coloca todos os números depois da vírgula. Assim, o número 123.45 é interpretado pelo seu computador como 0.12345×10^3 e representado apenas pelos números depois da vírgula (mantissa) e o expoente da potência, ou seja, 12345 e 3.

Número	Notação Científica	Mantissa	Base	Expoente
1000000000	0.1×10^{10} ou 1E9	1	10	10
123000	0.123×10^6 ou 1.23E5	123	10	6
456.78	0.45678×10^3 ou 4.5678E2	45678	10	3
0.00123	0.123×10^{-2} ou 1.23E−3	123	10	−2

Na verdade, o computador não guarda os números exatamente assim… Antes de fazer as operações, ele interpreta a base decimal (que nós normalmente usamos), e converte para uma base binária. Mas isso é assunto de outros cursos...

Bom, agora eu consigo representar qualquer número real no computador? Não exatamente… Toda máquina tem um limite de memória, um limite de memória implica em um limite na sua representação dos números. Assim, você não consegue representar números reais que ultrapassam um certo tamanho no seu computador. Com isso em mente, vamos apresentar os dois tipos de variáveis que podem ser usados para representar números reais em C.

Para representar números com menor precisão, usamos uma variável do tipo **float**. Uma variável do tipo float ocupa 4 bytes (32 bits) de memória e pode representar números que, em módulo, variam de 3.4×10^{38} (muito grandes!) a 3.4×10^{-38} (muito pequenos!).

Já para representar números maiores (ou muito pequenos), usamos uma variável do tipo **double**, que ocupa 8 bytes (64 bits) de memória e pode representar números que variam de 1.7×10^{308} até 1.7×10^{-308}.

Para declarar uma variável desse tipo, sigo o mesmo processo usado para declarar variáveis do tipo int, porém substituindo por **float** ou **double**:

```
int main(){
    float pequeno;
    double grande;

    pequeno = 0.12345678;
    grande = 123456789.987654321;
```

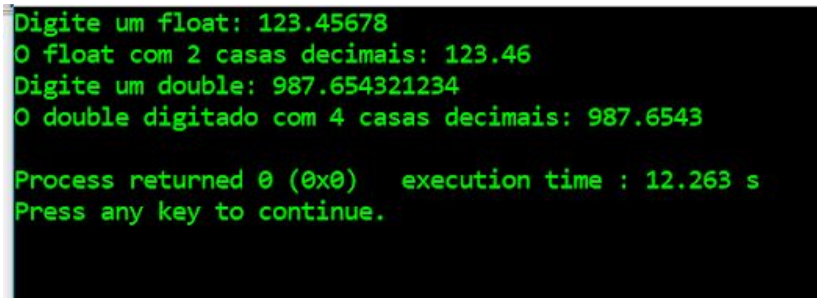
Para ler ou imprimir um número real, o processo também é parecido com o usado para números inteiros, porém alterando um pouco o comando.

Se estamos trabalhando com um **float**, usamos **%f**, e se estamos trabalhando com um **double**, usamos um **%lf** (double = long float).

Note que, em ambos os casos, o número que você digitou foi impresso com mais casas decimais do que a entrada. Como você altera isso? Para definir com quantas casas decimais você quer imprimir um número real, você coloca dois números depois do % separados por ponto. O primeiro número representa o número de dígitos antes da vírgula e o segundo, o

número de dígitos depois da vírgula. É comum controlarmos apenas o número de dígitos depois da vírgula.

```c
int main(){
   float pequeno;
   double grande;

   printf("Digite um float: ");
   scanf("%f",&pequeno);
   printf("O float digitado foi %.2f\n",pequeno); //representacao com apenas 2 casas decimais

   printf("Digite um double: ");
   scanf("%lf",&grande);
   printf("O double digitado foi %.4lf\n",grande); //representacao com 4 casas decimais

   return 0;
}
```



Perceba que se pedirmos para imprimir *muitas* casas decimais de um float, os dígitos menos significativos assumem valores que não definimos em parte nenhuma do código. Isso acontece porque a mantissa tem um limite de memória. Para começar a ver esse efeito em doubles, precisamos mandar imprimir ainda mais casas decimais.

Agora vamos ver um pouco como funcionam as operações com números reais. Pode parecer trivial, mas não é. Até agora nós só trabalhamos com números inteiros, então nunca tivemos que nos preocupar com os tipos das variáveis que estão sendo usadas nas contas, mas ao trabalhar com números reais, esse cuidado é necessário.

Basicamente, o resultado da operação depende não só dos valores que estão sendo usados, mas também dos tipos das variáveis que estão sendo usadas na operação e da variável que é usada para armazenar esse resultado. Isso ficará mais claro a seguir, mas só para dar uma ideia rápida:

- **inteiro** com **inteiro**→ **inteiro**
- **inteiro** com **real** → **real**
- **float** com **real** → **real**

Ou seja, quando um dos operandos for um real (**float** ou **double**), a operação é tratada pelo compilador como real. Veja o exemplo a seguir:

```c
int main(){

    printf("Divisao de inteiros: %d\n", 5 / 2); //divisao de inteiros retorna um valor inteiro
    printf("Divisao de floats: %f\n", 5.0 / 2.0); //divisao de floats retorna um float
    printf("Divisao de inteiro e float: %f\n", 5 / 2.0); //divisao de um inteiro e um float retorna um float

    int resi = 5.0 / 2.0;
    printf("Divisao de floats em variavel int: %d\n", resi); //divisao de floats em variavel int da ruim

    float resf = 5 / 2;
    printf("Divisao de int em variavel float: %f\n", resf); //divisao de ints em variavel float retorna o valor inteiro
                                                            //em um float

    return 0;
}
```

```
Divisao de inteiro e float: 2.500000
Divisao de floats em variavel int: 2
Divisao de int em variavel float: 2.000000

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

A partir desse exemplo, vamos entender melhor o que acontece. Quando você usa duas variáveis de tipos diferentes para fazer uma conta, o seu compilador transforma a “menos complexa” na “mais complexa” antes de realizar a conta. Isso acontece ao dividir um inteiro por um float, primeiro o seu compilador interpreta o inteiro como float e realiza a operação como se fossem dois floats ($5 / 2.0 \rightarrow 5.0 / 2.0 = 2.5$).

Agora, quando você faz uma conta com variáveis diferentes da variável que recebe esse resultado (como foi feito com as variáveis **resi** e **resf**), o compilador interpreta isso de um jeito diferente. Primeiro ele faz a conta com a variável do tipo que recebeu e depois a transforma no tipo que recebe o resultado. Isso fica bem visível no exemplo de divisão de inteiros armazenados em float ($resf = 5 / 2$). Primeiro o computador faz a divisão de 5/2 como inteiros, retornando o valor 2 e depois transforma em float, e atribuindo o valor de 2.00000 para a variável **resf**.

Como evitar isso? Uma boa prática é sempre que quiser trabalhar com reais, representar os números com uma casa decimal, mesmo se forem inteiros (ao invés de escrever 5, escreva 5.0).

A imagem abaixo exemplifica as diferentes operações entre reais e inteiros que podem ser realizadas:

```
int i,j;
float y;

i = 5 / 3; /* divisão inteira e o resultado é 1 (5 e 3 são inteiros) */
y = 5 / 3; /* divisão inteira e o resultado é 2.0 (y é real) */
y = 5.0 / 2; /* divisão tem como resultado 2.5 (o numerador é real) */
y = 5 / 2.0; /* divisão tem como resultado 2.5 (o denominador é real) */

y = i / 2; /* divisão inteira (i e 2 são inteiros) */
y = i / 2.0; /* divisão em ponto flutuante (denominador real) */
y = i / j; /* divisão inteira (i e j são inteiros) */
y = (1.0 * i) / j; /* divisão em ponto flutuante (numerador real) */
y = 1.0 * (i / j); /* divisão inteira (i e j são inteiros) */
i = y / 2; /* parte inteira da divisão em i (divisão real, mas i é inteiro) */
```

Parte 2 - Leitura de arquivos

Até agora nós só trabalhamos com entradas que vêm do teclado, mas e se quisermos ler o conteúdo de um arquivo de texto do nosso computador? Isso é o que aprenderemos a seguir.

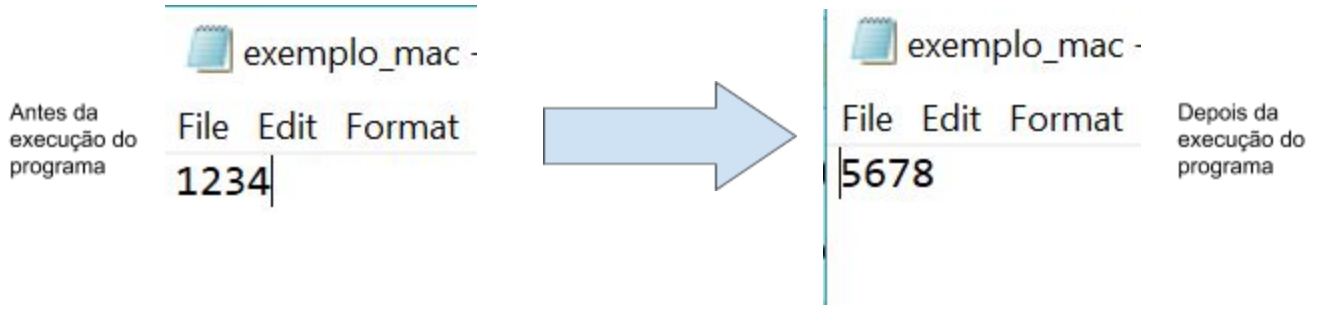
Vamos seguir uma ordem de passos:

1. Abrir o arquivo (**fopen**)
2. Realizar operações de leitura (**fscanf**) e escrita (**fprintf**) com o arquivo
3. Fechar o arquivo (**fclose**)

Veja os exemplos a seguir:

```c
int main(){
    FILE *entrada = fopen("exemplo_mac.txt", "r"); /*r indica que estamos no
                                                    modo read, de leitura*/

    int i;

    fscanf(entrada, "%d", &i); /*leio o conteudo do meu arquivo*/
    printf("%d", i);

    FILE *saida = fopen("exemplo_mac.txt", "w"); /*indica que estamos no modo de
                                                  escrever, ou seja, posso alterar
                                                  esse arquivo*/

    fprintf(saida, "%d", 5678);

    fclose(entrada);
    fclose(saida);

    return 0;
}
```

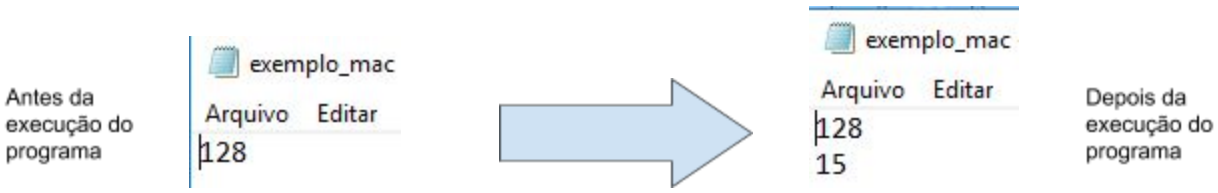
```
1234
Process returned 0 (0x0)   execution time : 0.048 s
Press any key to continue.
```



No exemplo acima, nós abrimos o arquivo no modo **"r"** (leitura), como a variável **entrada**, e usamos essa variável para ler o seu conteúdo, que é "1234". Depois, para conseguirmos escrever algo, precisamos abrí-lo novamente, dessa vez no modo **"w"** (escrita). Por fins didáticos, achamos melhor criar outra variável, **saída**. Com essa variável eu consigo realizar operações de escrita no arquivo (*exemplo_mac.txt*), porém a escrita substituirá o conteúdo anterior do arquivo (overwrite).

```c
#include <stdio.h>

int main(){
    int numero;
    /* Abertura do arquivo "exemplo_mac.txt" para leitura e escrita */
    FILE *arquivo = fopen("exemplo_mac.txt", "a+");

    /* Leitura do arquivo */
    fscanf(arquivo, "%d", &numero);
    printf("O numero contido no arquivo eh: %d\n", numero);

    /* Escrita no arquivo */
    printf("Digite um numero para ser inserido no arquivo: ");
    scanf("%d", &numero);
    fprintf(arquivo, "\n%d", numero);

    /* Fechamento do arquivo */
    fclose(arquivo);

    return 0;
}
```

```
O numero contido no arquivo eh: 128
Digite um numero para ser inserido no arquivo: 15

Process returned 0 (0x0)   execution time : 5.201 s
Press any key to continue.
```



Já no segundo exemplo, utilizamos um único tipo de variável, **arquivo**, que serve tanto para leitura quanto para escrita, utilizando o modo **"a+"** como parâmetro da função **fopen** (se estiverem confusos em relação à esses modos de leitura/escrita,vejam a explicação desse site). Em seguida, realizamos a leitura do número contido no arquivo, e o imprimimos na tela (no caso, o número é 128). Por último, escrevemos um número que o usuário digitou no nosso arquivo "exemplo_mac.txt", e chamamos a função **fclose** para encerrar a escrita/leitura.

Lembrem-se de sempre fechar os arquivos quando terminarem de ler ou escrever algo neles. Isso é importante porque, em alguns casos, outros programas não poderão manipular arquivos enquanto eles ainda estiverem abertos em um código nosso.

Caso queiram saber mais sobre operações de leitura/escrita com arquivos:

https://www.ime.usp.br/~elo/IntroducaoComputacao/Manipulacao%20de%20arquivo.htm